Basketball Detection: From Images to Videos

Justin Chang
Stanford University
jchang13@stanford.edu

Pin-Hsuan Tseng
Stanford University
tseng98@stanford.edu

Abstract

This project focuses on detecting basketballs in a wide range of images using deep learning models. We tuned and trained with three object detection models: Faster R-CNN, RetinaNet, and YOLO, and evaluated their performance using Intersection over Union (IoU) and mean Average Precision (mAP) metrics. Our findings show that RetinaNet outperforms the other models, achieving a mean Average Precision (mAP) of 0.977. The model detected really well in diverse settings with some false positive exceptions misclassifying shoulders and faces. We further utilized the RetinaNet model to analyze frames of video footage featuring 3-point shot attempts, aiming to track the basketball's position. We noticed a trend in the model facing detecting difficulties in scenarios with motion blur, crowded scenes, and low-resolution images.

1. Introduction

Sports is a universal hobby and activity that unites people across different countries, background, and age. Especially in the digital television era, fans across the world can tune in and watch games live across a screen. With the rapid expansion of recorded sports data and footage on the internet, data-driven approaches have become popular in the recent years to help analyze game statistics. One such popular techniques is to use computer vision with deep learning to help detect balls in a fast-paced game like basketball. With an accurate model of ball detection, it further unlocks other capabilities like trajectory estimation and basketball shot success prediction. One immediate benefit of applying ball detection techniques is an automated method for determining legal vs. foul play in dubious and fast game scenarios. Moreover, these computer vision approaches can also help demonstrate trends in players' sports performance that typical images or videos cannot show. Therefore, players hugely benefit by correcting their play styles and pose from understanding the outputs of vision-based models.

However, directly using feed from live games to detect ball locations can be a tricky task. Different camera angles with noisy audience background can make the training process very difficult. Therefore, this project focuses on using a highly diverse image dataset to train a basketball detection model and seeing it's effectiveness on game video footage. Hopefully by first having an accurate representation of what a basketball is, the trained model is capable of detecting the target in sequential frames. The input of our algorithm is basketball images in various settings, typically snippets of game footage, household settings, solo ball pictures, and player head shots. We then trained a FasterR-CNN, RetinaNet, and Yolo to detect for ball positions. The output is a bounding box around the ball location with a probabilistic confidence score of the object's presence. An evaluation of the success and failure cases of the best model was further conducted to examine the effectiveness of our approach.

2. Related Work

For ball detection tasks, common approaches fall into three categories: physics-based, computational imaging, and learning-based. For example in physics-based approaches, authors propose to first record blue-painted basketball videos with a black background and apply a global fixed color threshold for image filtering [9]. Then, they apply kinematics to calculate the ball velocity, throw angle, and ultimately position. This attempt demonstrates an advantage for modeling ball kinematics, but is restricted to controlled environments where noise is kept limited. Since this project aims to detect balls with a variety of background noise, the assumption of ball color and low noise background does not hold.

Another attempt for ball detection is with color information and Hough transform [16]. Since balls are round in shape, the Hough transform is applicable for detecting regular curves and lines while relatively unaffected by noise. This method is excellent in generic circle detections, but it doesn't have the flexibility for differentiating details between circular shapes, such as faces versus balls. Other authors at [3] have addressed this issue by first applying a circle Hough transform to see where the ball is and uses a neural classifier to detect for false positives. This approach makes it much more viable in sports settings where the color and shape of the targets are more distinct and easier to train with. Others have also decided to focus on the countour and centroid method for ball tracking [1].

Lastly, learning based methods such as the Region-based Fully Convolutional Networks (R-FCN) have gain popularity for object detection. For example, authors at [4] combine Online Hard Example Mining and Soft-NMS with R-FCN results to yield better detection accuracy. Cascading network input/outputs with other networks demonstrates to be effective in detecting and tracking basketballs from public live datasets. Other authors have also gravitated towards specializing in detection with fast inference times. Authors in [12] use a coarse regional proposal for the bounding box of a soccerball, then feed into a light-weight convolutional neural network to finalize on the ball location details. This method allows for near real-time detection, which is well equipped on robotic systems that allows for rapid decision-making using limited computation and power budgets. Authors in [17] also directly uses a bidirectional LSTM to directly do trajectory prediction. Some also use a more traditional RNN for the same task [10], while others specialize in more multi-object ball tracking [6]. Yolo has also been a popular model for sports tracking implementation due to its fast inference speed [15].

3. Methods

To enable our approach to detect ball targets in diverse settings, we chose to use a learning-based method. We approached our problem in three stages: Algorithm, Optimization, and Metric Selection.

3.1. Model Selection

Since we were interested in detecting ball position, this task becomes an object detection problem. Out of all the object detection algorithms, this project focuses on 1) Faster R-CNN, 2) RetinaNet 3) Yolo. All implementations used Pytorch's pre-defined model architectures.

3.1.1 Faster R-CNN

Faster R-CNN (Region-based Convolutional Neural Networks) is a deep learning method that aims to be an efficient and accurate detection algorithm [8]. It relies on a CNN as a backbone that extracts features from a given input image and cascades with a small network that outputs a bounding box (pixel x,y min and pixel x,y max) and also the probability of that item's within the box region. It's then fed into an ROI Pooling layer which standardizes the region dimensions regardless of the different aspect ratios, to finally be put into a fully connected layer that classifies the object category or the background.

3.1.2 RetinaNet

RetinaNet is a state of the art single shot detection that relies on a feature pyramid network backbone with two subnets for 1) classifying anchor box and 2) comparing anchor boxes to the ground truth [5]. In order to address class imbalance (background are more prevalent than foreground) it introduces focal loss which effectively focuses training on misclassified examples. RetinaNet is also a one-stage detector that performs object localization and classification in a single pass, making it suitable for fast training and inference time. This project incorporated RetinaNet for its advantage in speed whilst maintaining comparable accuracy.

3.1.3 YOLO

YOLO (You Only Look Once) is also single shot detection algorithm that optimizes for speed with fair accuracy [7]. It first discretizes the image to a grid and applies a bounding box with confidence scheme around the grid. It also produces a class probability map of the cells in the grid. Then it combines the boxes, confidence score, and probability to yield the object detection.

3.1.4 Finetuning

Because all three algorithms are catered to generic object detections that's not limited to just basketballs, we modified the box predictor dimensions to adapt to our custom dataset. The box predictor now has only two classes, either a 1 for a basketball being present and 0 for background. Aside from training from scratch, we also experimented with transfer learning using pretrained models. Pretrained models used the default weights, which are models that have been trained under COCO - an object detection, segmentation, and captioning dataset with over 200,000 labeled images with 80 object categories. To compare the different models' variation efficiency, the FasterR-CNN with no pretrained weights was chosen as the baseline.

3.2. Parameter Optimization

3.2.1 Optimizers

Once the architectures were decided, we experimented with ADAM, RMSProp, and SGD optimizers. For ADAM, we experimented with learning rates between $0.0001 - 0.001$ and weight decay between $0 - 0.001$. The learning rate controls how big of a step towards the negative gradient for the next iteration and the weight decay controls how much generalization our models should behave to prevent over-fitting to training data. In addition, RMSProp had momentum values between $0.5 - 0.9$ and alpha values between $0.9 - 0.99$. Moreover, we used damping values between $0 - 0.1$. 20 random sampling of the hyperparameters were used and ran with 400 training images. We used a small training image set to quickly cycle through different optimizers under different settings and chose the one with the lowest loss value.

3.2.2 Hyper-parameter Tuning

Once an optimal optimizer is determined, we performed a grid search on hyperparameters to better tune our model's performance. Previously, during optimizer search a broad range of hyperparameters were selected. In this step a grid search was selected for a more in-detail evaluation of hyperparameters, hoping to hone in on finding the best performing model ready to train for longer epochs.

3.3. Metrics

3.3.1 Intersection over Union (IoU)

To quantify the model performance, we use two metrics: Intersection over Union (IoU) and Average Precision (AP).

IoU is a measure of the overlap between the predicted bounding box and the ground truth bounding box. It is calculated as follows:

$$IoU = \frac{\text{Area of Intersection between two boxes}}{\text{Area of Union between two boxes}} \tag{1}$$

The IoU score ranges from 0 to 1, where 1 indicates a perfect overlap. As a demonstration, Figure 1 shows an IoU with our dataset. The blue box refers to the ground truth and the red dotted box is a proposed region. Both red boxes have similar area of unions, but the smaller box has a smaller intersection and therefore a smaller IoU.

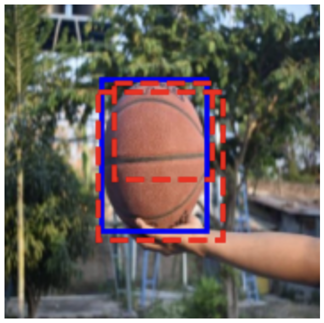

Figure 1. Proposed Bounding Boxes

However, using only IoU as a metric has limitations. 1) What threshold should we set to identify as a real detection? 2) How should multiple detections of the same object in the image be handle? To address these issues, Average Precision (AP) was used as a more holistic and complete metric for accuracy. The rest of the project compares primarily mean AP values and uses IoU as a secondary metric.

3.3.2 Average Precision (AP)

Average Precision works as the following: a bounding box is first produced by the network and the model's output probability score is compared with a certain threshold. Since there are only two classes, the probability indicates how likely this bounding box contains a basketball. Then, the values of the confusion matrix are computed as shown in Table 1.

	Prediction (+)	**Prediction (-)**
GT (+)	TP (True Positive)	FN (False Negative)
GT (-)	FP (False Positive)	TN (True Negative)

Table 1. Confusion Matrix

Precision is then defined as the ratio of correct predictions to total predictions, which measures the accuracy of the model's predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

Similarly, recall is defined as the ratio of correct predictions to ground truth (GT) instances, which effectively measures the model's ability to detect all instances.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

There's a trade-off relationship between precision and recall: increasing the threshold improves precision but decreases the recall value. Therefore, to evaluate a model's prediction performance, AP is calculated in Equation 4 as the area under the precision-recall curve. The range of AP is from 0 to 1 where 1 indicates a theoretically perfect model.

$$\text{AP} = \int_0^1 p(r)dr \tag{4}$$

mean Average Precision (mAP) is then calculated as the average of AP values across all classes. Since in our case we only have one class, the mAP is the same as AP.

4. Dataset and Features

Our dataset is sourced from Roboflow's open dataset, found in this link. [2] This dataset comprises of 17,505 training images, 2,738 validation images, and 1,158 test images. The dataset collects a diverse and well-annotated set of basketball images, including those from various indoor and outdoor venues, under different lighting conditions, and from different angles. Most images come from different national leagues, game scenarios, player portraits, and household settings. A distinctive feature of this dataset is the significant variation in the color, size, occlusion blur, and noise of the basketballs. These variations allow our model to learn more features during training and generalize better to different scenarios during testing. Figure 2 shows an example of a few images in the dataset.

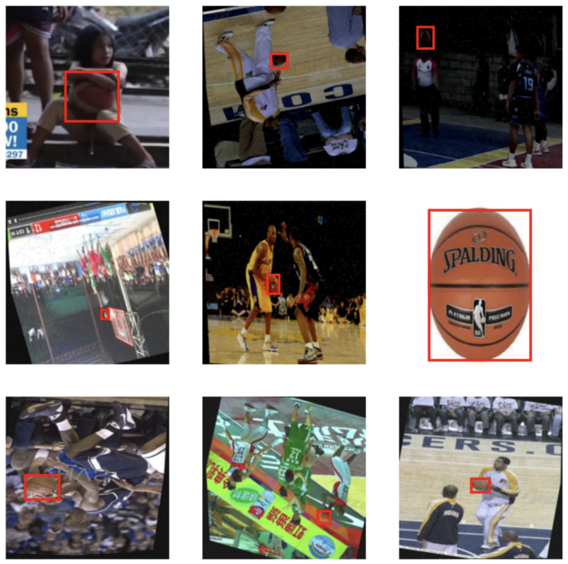

Figure 2. Sample Images in the Dataset

For data pre-processing, all collected materials in this dataset are uniformly resized to 640 x 640 pixels. As for data augmentation, images within the dataset come flipped, rotated within 15°, and sheared. The images also undergo blurring, color saturation (between -31% and +31%), and brightness modification (between -20% and +20%).

4.1. Video Dataset

Aside from the public dataset, this project also experimented using the best trained model to evaluate its detection efficiency on video footage. The dataset consists of recorded 3-point shot attempts in NBA games [14] [11] [13]. Each clip includes 32 frames of wide-shot images with a resolution of 1280 x 720 pixels. To fit the same image dimension as Roboflow's dataset, we pre-processed each frame by resizing it to 640 x 640. Then the dataset is fed into the model in the same way as the previous dataset.

5. Experiments and Results

5.1. Tuning

Once the architectures were selected, the first thing we experimented with was the optimizer. Running 3 architectures with 3 optimizers at both pre-trained and not trained settings would yield 18 different runs. Because this is only the initial effort in tuning our results, we opted to run using only our baseline's architecture with 20 different parameters for 400 images. The results are shown in Table 2.

The ADAM optimizer using a learning rate of $2.85e-5$ with weight decay of $3.4e-4$ demonstrated the lowest loss value. This made intuitive sense because of ADAM's capability for adaptive learning rates compared to Stochastic Gradient Descent and also over RMSProp due to its inclusion of the first moment. Therefore, the rest of the project uses ADAM on all the models for a more unbiased comparison.

Setup	Loss Values
ADAM pre-trained	**0.1498**
ADAM not-trained	0.232
SGD pre-trained	0.153
SGD not-trained	0.232
RMSProp pre-trained	0.1514
RMSProp not-trained	0.222

Table 2. Optimizer Search with FasterR-CNN

Next, we performed a more in-detail parameter search of the hyperparameters. We ran for 1 epoch with 3 learning rates and 3 weight decays on our models. As an example, Table 3 shows the hyperparameters we tested on Yolo using grid search with LR referring to learning rate and WD as weight decay. The values shown are mAP values instead of loss because we wanted to use the same metric as our final comparisons for the sake of consistency. The experiment showed that having the highest learning rate demonstrated the highest mAP values. This intuitively also makes sense because a faster learning rate usually helps the solver converge to optimal faster with a bigger step towards the negative gradient. Interestingly, the middle value for weight decay showed best performance. This could be explained by striking a balance over-fitting versus over-generalization: too low of a weight decay could cause an overfit while too high of a value makes it over-generalize and hard to train the model.

When we applied the same grid hyperparameter search for the Faster R-CNN and RetinaNet, both models yielded nearly 0 mAP values during validation. The IoU values also wavered up and down without a clear sign of improvement. It became evident that the model wasn't learning and converging with the same parameters from Table 3. This could be explained by the disparity in architecture with our models, as the optimal hyper-parameters are tied to the optimizer but also the diversity in model backbones. Therefore, we implemented a similar search as in Table 2 with an epoch of 1 and still found the learning rate of $2.85e-5$ and weight decay of $3.4e-4$ performed best.

	LR 1E-3	LR 1E-4	LR 1E-5
WD 5E-3	0.54	0.296	0.141
WD 1E-4	**0.552**	0.281	0.142
WD 5E-4	0.523	0.315	0.198

Table 3. Hyperparameter Search with Yolo

5.2. Training

We then ran the three models for 5 epochs of the training data. As an example, Figure 3 shows RetinaNet's loss curves versus iterations. Due to GPU RAM limitations, each iterations takes in 32 training images and 16 validation images. After each iteration, the model's parameters are backward propagated and updated. Initially, the no-trained loss demonstrates higher values than pre-trained because it's learning from scratch. Because it's learning from near zero initialization, the training and validation loss decreases rapidly as seen from iterations from 0 to 500. However, the pre-trained models began to stagnate learning after 500 iterations. The high fluctuations in the training and validation values are due to the relatively small batch size. Regardless, the validation loss for both models shows an overall decrease with iterations, indicating an increase in model's performance over time.

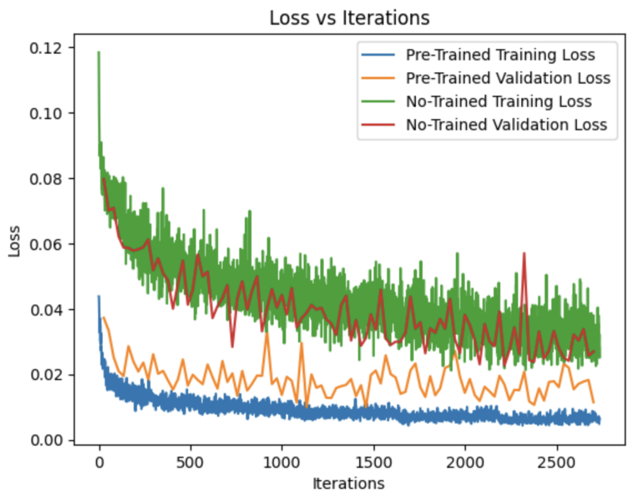

Figure 3. Loss Curves with Trained vs. Pre-trained

Aside from loss curves, the IoU and mAP values are also plotted in Figure 4. At each iteration, a prediction with a class score and bounding box is produced. The IoU values of the validation images are computed using the predicted vs. ground truth bounding box. The two boxes then gets appended to the model's history of seen data, which the mAP algorithm uses to compute a value. Figure 4 shows both the IoU and mAP values increasing, indicating that the model is improving with more training. Similar to the loss curves, the pre-trained models show better mAP/IoU performance than the no-trained models. Because this behavior is evident amongst all three model architectures, we proceeded to compare the architectures only with their pre-trained versions against the baseline (FasterR-CNN Notrained).

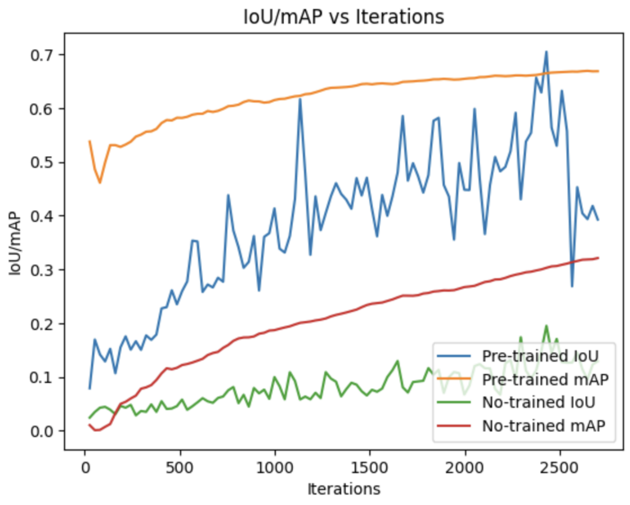

Figure 4. IoU/mAP with Trained vs. Pre-trained

5.3. Evaluation

5.3.1 Model Evaluation

Lastly, the transfer-learned models are then compared with one another along with the baseline. The models performed prediction based on test images with a outputs of bounding boxes and class scores. During testing, we noticed that the model outputs a lot of potential bounding boxes of where the basketball is. However, not all probability scores of these boxes are high. Therefore, we set a score threshold of 0.5 during evaluation to only account for boxes that the model deems as a higher ball chance than background. Figure 5 shows the difference in evaluation before and after the threshold.

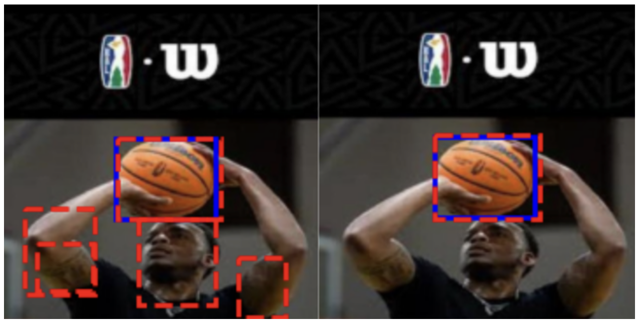

Figure 5. Left: Bounding Box Prior to Thresholding; Right: Bounding Box Post-Thresholding

With an applied score threshold, the mAP (referred as map50 because of the 0.5 detection) values could be calculated. The results are shown in Table 4 with RetinaNet as the best performer, achieving up to 0.977 accuracy. All three pretrained models demonstrated better performance than the baseline.

Figure 6 shows a series of sample outputs for our best

Model	mAP50
Baseline (Faster R-CNN Notrained)	0.919
FasterR-CNN (Pretrained)	0.935
RetinaNet (Pretrained)	**0.977**
Yolo (Pretrained)	0.963

Table 4. Model Evaluation Results

model's detection. Overall, the model performed really well in basketball detection. The top row demonstrates example cases where the detection is accurate: in-game scenarios, multi-target shots, and also solo scenes. In almost all the cases during validation the model was able to really accurately find the ball position.

However, there are a few scenarios where the model outputs false positives. As seen in the bottom row, hands, shoulders, and faces sometimes gets misclassified as basketballs. Based off of these false positive observations, it could be hypothesized that the model yields a positive output based off of color, shape, and texture. Since most training inputs have basketballs as dark brown and circular, it's easy for the model to heavily rely on those features. Aside from these few wrong examples, the model generally performed well in a high diversity of basketball images.

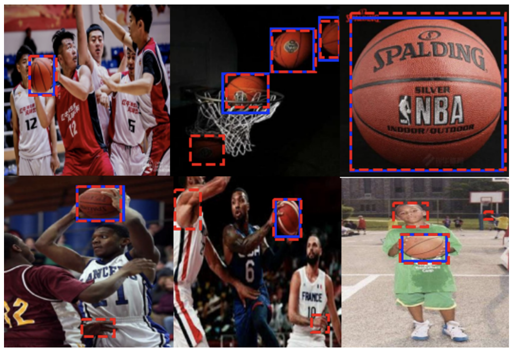
Figure 6. Qualitative Results

5.4. Applying RetinaNet to Video Footage

With the model's high performing result using our dataset, we were curious to see its effect on game video footage. The trained RetinaNet model was then applied on five short video clips, each demonstrating a 3-point shot attempt. The detection results of the RetinaNet model on the five clips are shown in figures 7 - 9.

Our observations indicate that when the confidence score was set to 0.5, the model scarcely detected any basketballs in the frames. If we lowered the confidence score to 0.1, the model did detect some basketballs but it comes as the cost of an increase in false positives. As an example, Figure 8 shows the basketball being detected where Figure 7 wasn't able to. Now with a lowered threshold, some players' heads became wrongly detected.

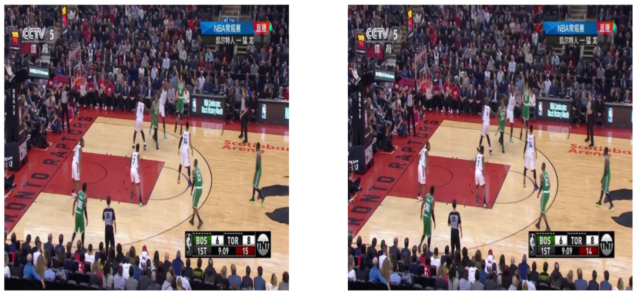
Figure 7. Detection Results - Confidence Score 0.5

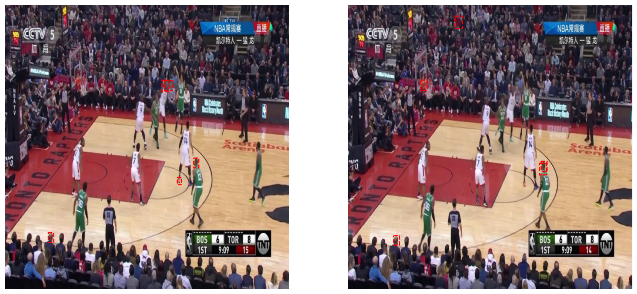
Figure 8. Detection Results - Confidence Score 0.1

In some scenarios where the ball is positioned with a highly contrasting background, similar to Figure 9's white floor, the model was capable of detecting the ball position. However, these cases were not the majority of the test set. The model overall performed subpar as compared to the datasets shown in Figure 6.

The poor detection behavior of the video could be explained with a few reasons. First, the training dataset focused only on individuals or a few number of people in the foreground. It lacked datasets with a large and noisy crowd background. Therefore, when presented with images with a large audience, our model struggled to detect where the basketball is. As a result, the model often misclassified human heads from the audience as the target.

Second, the raw video footage were designed to capture the whole playing field, so the camera tends to zoom out with few close-up shots. During data pre-processing, we reduced the resolution of the images to match the original public dataset dimension. This resizing caused the targets to appear blurry, hence causing the model to have a hard time detecting ball location.

Third, the videos were sampled with 32 frames per second. Since the ball is in motion, not every frame can detect a clear basketball in a fast-paced game. This motion blur further distorted the shape and clarity of the basketball, which hindered the model's prediction performance.

6. Conclusion and Future Work

In this project, we trained different object detection models to specialize in basketball detections. We conducted ex-

Figure 9. High Contrast between the Ball and Floor

periments with three distinct models: Faster R-CNN, RetinaNet, and Yolo. These models were tuned with finding an optimal optimizer along with its corresponding hyperparameters. The models were further evaluated using IoU and mAP metrics to evaluate their performance in prediction accuracy. By including a class score threshold, all models were fairly capable of detecting ball location. Our findings revealed that RetinaNet surpassed the other models, achieving a mAP of 0.977. Despite its overall effectiveness, the model occasionally misclassified non-basketball objects like hands, shoulders, and faces, which suggests a heavy reliance on features such as color, shape, and texture that are typical of basketballs.

Furthermore, we applied our best model RetinaNet to video footage of 3-point shot attempts. When applying to video footage, the threshold needed to be lowered for any ball detections. As a result, false positives started to arise with player and crowd heads as wrongful detections. The results from video footage were less promising than anticipated due to the lower-resolution of images and higher ball motion blur. In scenarios with high contrasting background images, the model was able to make detections because these were similar environments the training dataset included.

Looking ahead, an intriguing extension to this project would be to track the basketball within a video and predict the success of a shot. Both tracking and shot success prediction require a model that's capable of processing temporal information and a comprehension of the game context. Model architectures like LSTM, Vision Transformers, or 3D CNN are suitable candidates. The effort in this project could serve as a feature extractor and be incorporated into these larger models which are specialized to predict the outcomes of player shots. Such models could be more advantageous for coaches and players by providing new sports metrics that better help analyze the game whilst enhancing players' performance.

7. Contributions & Acknowledgements

Justin contributed to the majority of the writing of the paper. He also performed the optimizer tuning and the five epoch training on RetinaNet and FasterR-CNN. He also worked on providing the experiment plots along with writing the qualitative and quantitative findings of the results.

Pin-Hsuan was responsible for developing the core structure of the code, creating the custom dataset class, and implementing the training loop. He conducted the hyperparameter sweep for the FasterR-CNN and YOLO models, and carried out the five-epoch training on YOLO. He also wrote the code to test the performance of the models on the test dataset, evaluated the model on video footage, and documented the qualitative findings of the results on video footage.

References

[1] S. S. Ali Shah, M. A. Khalil, S. I. Shah, and U. S. Khan. Ball detection and tracking through image processing using embedded systems. In *2018 IEEE 21st International Multi-Topic Conference (INMIC)*, pages 1–5, 2018.

[2] Biomechanics. Basketball annotation training dataset. `https://universe.roboflow.com/biomechanics/basketball-annotation-training`, mar 2024. visited on 2024-06-04.

[3] T. D'Orazio, C. Guaragnella, M. Leo, and A. Distante. A new algorithm for ball recognition using circle hough transform and neural classifier. *Pattern Recognition*, 37(3):393–408, 2004.

[4] Q. Liang, L. Mei, W. Wu, W. Sun, Y. Wang, and D. Zhang. Automatic basketball detection in sport video based on r-fcn and soft-nms. In *Proceedings of the 2019 4th International Conference on Automation, Control and Robotics Engineering*, CACRE2019, New York, NY, USA, 2019. Association for Computing Machinery.

[5] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, Los Alamitos, CA, USA, oct 2017. IEEE Computer Society.

[6] A. Milan, L. Leal-Taixé, I. D. Reid, S. Roth, and K. Schindler. Mot16: A benchmark for multi-object tracking. *ArXiv*, abs/1603.00831, 2016.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2016.

[8] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[9] S. Saraireh, A. Hassanat, M. Abu Altaieb, and H. Kilani. A new dataset method for biomechanical training model of the free throws shots in basketball using image processing technique. *Modern Applied Science*, 13:132, 01 2018.

[10] R. Shah and R. Romijnders. Applying deep learning to basketball trajectories, 2016.

[11] J. Tang, X. Shu, R. Yan, and L. Zhang. Coherence constrained graph lstm for group activity recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[12] M. Teimouri, M. H. Delavaran, and M. Rezaei. A real-time ball detection approach using convolutional neural networks. In S. Chalup, T. Niemueller, J. Suthakorn, and M.-A.

Williams, editors, *RoboCup 2019: Robot World Cup XXIII*, pages 323–336, Cham, 2019. Springer International Publishing.

[13] R. Yan, J. Tang, X. Shu, Z. Li, and Q. Tian. Participation-contributed temporal dynamic model for group activity recognition. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 1292–1300, 2018.

[14] R. Yan, L. Xie, J. Tang, X. Shu, and Q. Tian. Social adaptive module for weakly-supervised group activity recognition. *arXiv preprint arXiv:2007.09470*, 2020.

[15] Y. Yoon, H. Hwang, Y. Choi, M. Joo, H. Oh, I. Park, K.-H. Lee, and J.-H. Hwang. Analyzing basketball movements and pass relationships using realtime object tracking techniques based on deep learning. *IEEE Access*, 7:56564–56576, 2019.

[16] H. Zhang, Y. Wu, and F. Yang. Ball detection based on color information and hough transform. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 2, pages 393–397, 2009.

[17] Y. Zhao, R. Yang, G. Chevalier, R. C. Shah, and R. Romijnders. Applying deep bidirectional lstm and mixture density network for basketball trajectory prediction. *Optik*, 158:266–272, 2018.